Monitoring Data Archives for Grid Environments

Jason Lee, Dan Gunter, Martin Stoufer, Brian Tierney
Lawrence Berkeley National Laboratory

Abstract

Developers and users of high-performance distributed systems often observe performance problems such as unexpectedly low throughput or high latency. To determine the source of these performance problems, detailed end-to-end monitoring data from applications, networks, operating systems, and hardware must be correlated across time and space. Researchers need to be able to view and compare this very detailed monitoring data from a variety of angles. To solve this problem, we propose a relational monitoring data archive that is designed to efficiently handle high-volume streams of monitoring data. In this paper we present an instrumentation and event archive service that can be used to collect and aggregate detailed end-to-end monitoring information from distributed applications. We also show how the archive fits into the Global Grid Forum's "Grid Monitoring Architecture".

1.0 Introduction

Developers and users of high-performance distributed systems often observe unexpected performance problems. It can be difficult to track down the cause of these performance problems because of the complex and often indirect interactions between the many distributed system components. Bottlenecks can occur in any of the components through which the data flows: the applications, the operating systems, the device drivers, the network interfaces, and/or in network hardware such as switches and routers.

In previous work we have shown that detailed application monitoring is extremely useful for both performance analysis and application debugging [25][2][24]. Consider the use-case of monitoring some of the High Energy Physics (HEP) Grid projects [16][11][7] in a *Data Grid* environment. These projects, which will handle hundreds of terabytes of data, require detailed instrumentation data to understand and optimize their data transfers. For example, the user of a Grid File Replication service [3][27] notices that generating new replicas is taking much longer than it did last week. The user has no idea why performance has changed -- is it the network, disk, end host, GridFTP server, GridFTP client, or some other Grid middleware such as the authentication or authorization system?

To determine what changed, one needs to analyze monitoring data from hosts (CPU, memory, disk), networks (bandwidth, latency, route), and the FTP client and server programs. In addition to recent measurements, recent historical data (e.g.: two or three weeks) are needed to establish a performance baseline. Only with this context can one begin to understand the current data. The user needs a way to mine this large historical dataset to extract only data relevant to their performance problem. Current performance can then be analyzed and compared against a baseline drawn from previously archived information.

A relational database that supports SQL [20] is an excellent tool for this type of task. SQL provides a general and powerful language for extracting data. For example, with SQL we can do queries such as:

- find the average throughput for the past 100 runs
- return all events for application runs that coincided with reports of network errors
- return all events for application runs where the throughput dropped below 10 Mbits/sec and CPU load was over 90%
- return all host and network events during application runs that took over 30 minutes
- return all events for application runs that failed (reported an error or never completed) during the last week
- return all events for applications runs where the total execution time was more than 50% from the average time for the past month

Over the past two years the Global Grid Forum's Grid Performance Working Group [10] has worked to define the "Grid Monitoring Architecture" (GMA) [23], which describes the major components of a Grid monitoring system and their essential interactions. In this paper we show how the archive uses the GMA "producer" and "consumer" interfaces.

We also address the scalability issues inherent to aggregating monitoring data in a central archiving component. The archive must be able to easily handle high-speed bursts of instrumentation results, in order to avoid become a bottleneck precisely when the system is most loaded.

2.0 Related Work

There are several application monitoring systems, such as Pablo [17], AIMS [28], and Paradyn [14]. However these systems do not contain archival components. One of the first papers to discuss the use of relational databases for application monitoring was by Snodgrass [19], who developed a relational approach to monitoring complex systems by storing the information processed by a monitor into a historical database. The Global Grid Forum Relational Database Information Services research group is advocating the use of relational models for storing monitoring data, and this group has produced a number of documents, such as [4][8] and [6].

A current project that includes a monitoring archive is the *Prophesy* performance database [26]. Prophesy collects detailed pre-defined monitoring information at a function call level. The data is summarized in memory and sent to the archive when the program completes. In contrast, our system for analyzing distributed applications, called *NetLogger* (and described below), provides a toolkit for user-defined instrumentation at arbitrary granularity. Typically this generates far too much data to hold in memory, so the data must be streamed to a file or socket. This means that our archive architecture must handle much more data that the Prophesy system. In addition, Prophesy includes modeling and prediction components, where our system does not.

The Network Weather Service team is currently adding an archive to their system, and the data model we are using, described below, is derived from the NWS data model described in [21]. There are several other monitoring systems that are based on the Global Grid Forum's GMA, including CODE [18] and R-GMA [9]. R-GMA contains an archive component, but does not appear to be designed to handle large amounts of application monitoring data. Spitfire [13] is a web service front-end to relational databases, which could potentially be used for an event archive.

3.0 Monitoring Components

The system described in this paper has four main monitoring components: the *application instrumentation*, which produces the monitoring data; the *monitoring activation service*, which triggers instrumentation, collects the events, and sends them to the requested destinations; the *monitoring event receiver*, which consumes the monitoring data and converts the events to SQL records and writes them to a disk buffer; and the *archive feeder*, which loads the SQL records into an event archive database. These components are illustrated in Figure1. A previous paper focused on the first two components [12], while in this paper, we focus on the last two components.

In order for a monitoring system to be scalable and capable of handling large amounts of application monitoring event, none of the components can cause the pipeline to "block" while processing the data, as this could cause the application to block while trying to send the monitoring to the next component. In general we have found that performance analysis of distributed systems requires the generation of monitoring events before and after every I/O operation. This can generate huge amounts of monitoring data, and great care must be taken to deal with this data in an efficient and unobtrusive manner. Depending on the execution environment, potential bottlenecks exist on the network from the producer to consumer, and inserting events into the event archive database. To avoid blocking, the system must impedance-match

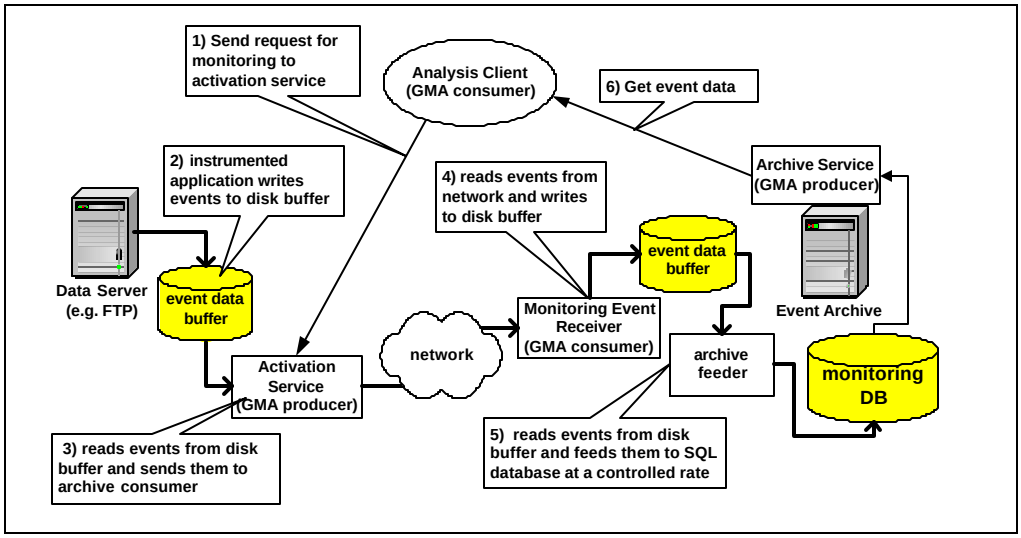

Figure 1: Monitoring Components

slow data “sinks” with fast data “sources” by buffering data to disk at all bottleneck locations, as shown in Figure 1. This is similar to the approach taken by the Kangaroo system for copying data files [22].

Of course, if the *sustained* data rate exceeds the maximum speed of the slowest component (e.g.: the network or database loading), then the disk buffers will eventually fill and the pipeline will block. However, in many application debugging and tuning scenarios, high monitoring data rates come in *bursts*, for example for the duration of a file transfer or the run of a single set of parameters, between which there is only low-frequency “background” monitoring such as CPU or network probes. In this environment, the slower components will not block the pipeline, but only add some latency as the data waits in a buffer for processing.

4.0 Previous Work

We now briefly described two previous components that this system is built upon, NetLogger and GMA.

4.1 NetLogger Toolkit

At Lawrence Berkeley National Lab we have developed the *NetLogger Toolkit* [25], which is designed to monitor, under actual operating conditions, the behavior of all the elements of the application-to-application communication path in order to determine exactly where time is spent within a complex system. Using NetLogger, distributed application components are modified to produce timestamped logs of “interesting” events at all the critical points of the distributed system. Events from each component are correlated, which allows one to characterize the performance of all aspects of the system and network in detail.

All the tools in the NetLogger Toolkit share a common log format, and assume the existence of accurate and synchronized system clocks. The NetLogger Toolkit itself consists of four components: an API and library of functions to simplify the generation of application-level event logs, a set of tools for collecting and sorting log files, an event archive system, and a tool for visualization and analysis of the log files. In order to instrument an application to produce event logs, the application developer inserts calls to the NetLogger API at all the critical points in the code, then links the application with the NetLogger library. All the tools in the NetLogger Toolkit share a common log format, and assume the existence of accurate and synchronized system clocks. We have found that for this type of distributed systems analysis, clock synchronization of 1 millisecond is required, and that the NTP [15] tools that ship with most Unix systems (e.g.: *ntpd*) can easily provide this level of synchronization.

We have found exploratory, visual analysis of the log event data to be the most useful means of determining the causes of performance anomalies. The NetLogger Visualization tool, *nlv*, has been developed to provide a flexible and interactive graphical representation of system-level and application-level events.

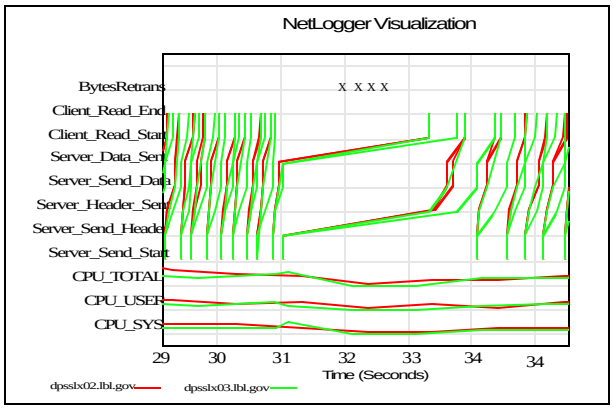

Figure 2: Sample NetLogger Results

Figure2 shows sample *nlv* results, using a remote data copy application. The events being monitored are shown on the Y axis, and time is on the X axis. CPU and TCP Retransmission data are logged along with application events. Each lifeline represents one block of data, and one can easily see that occasionally a large amount of time is spent between *Server_Send_Start* and *Client_Read_Start*, which is the network data transfer time. From this plot it is easy to see that these delays are due to TCP retransmission errors on the network.

We recently added to NetLogger an efficient self describing binary wire format, capable of generating 615,000 events per second [12]. This means that we can generate over 6000 events per second with only a 1 percent perturbation of the application.

NetLogger’s ability to correlate detailed application instrumentation data with host and network monitoring data has proven to be a very useful tuning and debugging technique for distributed application developers.

4.2 Grid Monitoring Architecture (GMA)

We have helped lead a Global Grid Forum (GGF) effort to defined a highly scalable architecture for Grid monitoring, called the *Grid Monitoring Architecture*, or GMA. This work has taken place in the GGF Performance and Information Area, which also has groups working to standardize the protocols and architectures for the management of a wide range of Grid monitoring information, including network monitoring.

The prime motivation of the GMA is the need to scalably handle dynamic performance information. In some models, such as the CORBA Event Service [5], all communication flows through a central component, which represents a potential bottleneck in distributed wide-area environments. In contrast, GMA performance monitoring data travels directly from the producers of the data to the consumers of the data. In this way, individual producer/consumer pairs can do "impedance matching" based on negotiated requirements, and the amount of data flowing through the system can be controlled in a precise and localized fashion. The design also allows for replication and reduction of event data at intermediate components acting as caches or filters.

In the GMA, the basic unit of monitoring data is called an *event*. An event is a named, timestamped, structure that may contain one or more items of data. This data may relate to one or more resources such as memory usage or network usage, or application-specific types of data like the amount of time it took to multiply two matrices. The component that makes the event data available is called a *producer*, and a component that requests or accepts event data is called a *consumer*. A *directory service* is used to publish what event data is available and which producer to contact to request it.

The GMA architecture supports both a subscription model and a request/response model. In the former case, event data is streamed over a persistent "channel" that is established with an initial request. In the latter case, one item of event data is returned per request.

The GMA architecture has only three components: the producer, consumer, and directory service. This means that only three interfaces are needed to provide interoperability, as illustrated in Figure3.

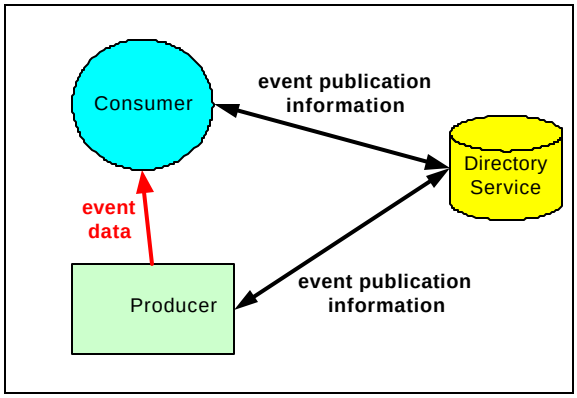

Figure 3: Grid Monitoring Architecture Components

The **directory service** contains only metadata about the performance events, and a mapping to their associated producers or consumers. In order to deliver high volumes of data and scale to many producers and consumers, the directory service is not responsible for the storage of event data itself.

A **consumer** is any program that requests and/or receives event data from a producer. In order to find a producer that provides desired events, the consumer can search the directory service. A consumer which passively accepts event data from a producer may register itself, and what events it is willing to accept, in the directory service.

A **producer** is an program that responds to consumer requests and/or sends event data to a consumer. A producer that accepts requests for event data will register itself and the events it is willing to provide in the directory service. In order to find a consumer that will accept events that it wishes to send, a producer can search the directory service.

5.0 Monitoring Event Receiver and Archive Feeder

The *monitoring event receiver*, shown in Figure1, reads monitoring events from the network, parses the events into a format that can be directly loaded into the database, and writes them to disk. The *archive feeder* then periodically loads the disk files into the database.

The database queries suffer performance degradation while new data is being loaded. Therefore, the feeder is paced to avoid locking out interactive queries during continual LOAD commands. This pacing can be configured to partially control the balance between load rate and query responsiveness.

The other control we have over the load rate vs. responsiveness trade-off is the size of the disk files. The monitoring event receiver creates disk files containing a pre-defined number of events. The larger the file, the longer the database will be busy (and the host CPU will be near 100% utilization) for each LOAD command. We ran a series of tests to determine the optimal size for the files, and found that the load rate is fairly constant for files with between 500 and 25,000 events. For the database we are using (mySQL), we have found that loading about 5000 events at a time maintains a good balance between storing sufficient data in a timely manner, while using the database is efficiently as possible.

Even with the incoming data reduced to a trickle, the database will not be able to execute SQL queries efficiently if the data model is inappropriate (or poorly implemented in the database tables). Our data model, described next, is simple but also performs well for common types of queries.

6.0 Data Model and Data Archive

Our data model, shown in Figure4, is based on NWS archive work [21]. It is very simple: we describe each *event* with a name, timestamp, "main" value, "target", program name, and a variable number of "secondary" string or numeric values. The target consists of a source and destination IP address, although the destination address may be NULL. There is a many-to-one relationship from events to event *types*, and events to event *targets*. Therefore these *entities* can be put into separate indexed tables to allow fast mapping and searching of events. Figure5 shows sample events for TCP throughput (from *iperf*) and application monitoring (from a *GridFTP* server) represented in this model.

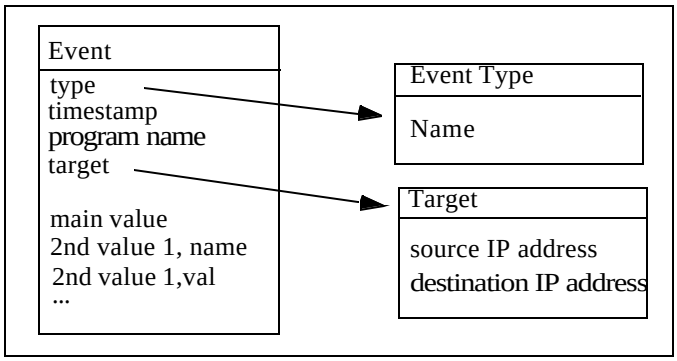

Figure 4: Event data model

We have optimized the actual database tables somewhat from the general model described above, both for database size and speed. These optimizations are based on common-sense and an intuition of the most frequent types of queries. For example, the "secondary" values are subdivided into two tables, one for strings and one for numbers, because storing both in the same table would waste space and slow down indexing. We are also using the event timestamp as the primary database index, permitting quick access to ranges of dates.

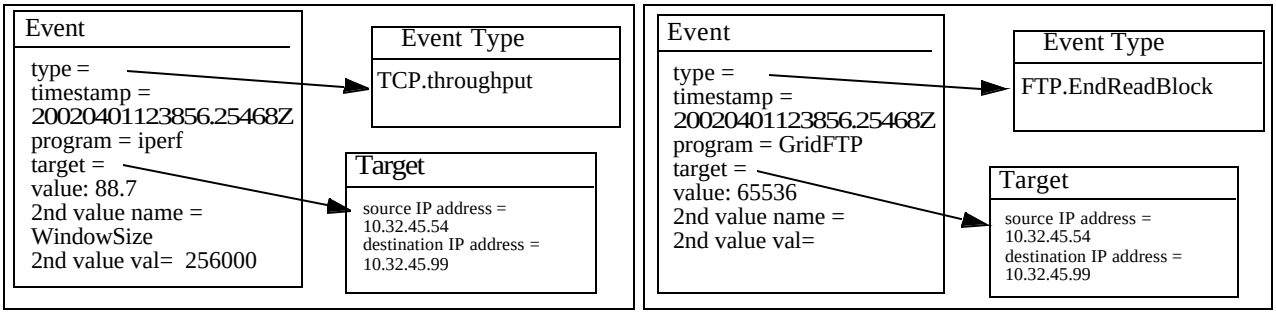

Figure 5: Example events (iperf and GridFTP)

7.0 Use of GMA

The GMA provides a common framework for structuring the interactions between the user and the activation service, event receiver, and archive service components. In GMA terms, the activation service is a producer, the event receiver is a consumer, and the archive service is a producer. When requesting activation from the activation service, or querying the archive service, the user takes the GMA role of a consumer.

To illustrate this, consider the process of sending results for a file transfer to the archive and then, some time after the transfer has finished, making a query to the archive that will use those results. To begin with, the user *subscribes* to the activation service, indicating that results should be sent not back to itself, but instead to the event receiver. The user is a *consumer* asking for events from the activation service *producer*, and directing the results to the event receiver *consumer*. If all goes well, the activation service will send monitoring data to the event receiver, and from there the data will move into the event archive. Either the user will know where the event archive is, or the event archive can be intelligent and register that it has events for this file transfer in a *directory service* that the user can search. Either way, the user will then contact the event archive directly, again acting as a *consumer* asking a *producer* for events. This time, the user may *query* the event archive, embedding for example some SQL statements into the request, and receive their desired information as a response.

Using the GMA interfaces, we can provide a coherent framework for the series of interactions necessary to activate, archive, and retrieve event data.

8.0 Sample Results

The following description demonstrates the power of SQL for analyzing monitoring data. Consider the case of unexplained periodic dips in network throughput. To understand the cause, we construct a query to find all events that happened during any time when the network throughput was below 1/5 of the average on that path. This query is simply a matter of extracting links with bandwidth measurements, which are already defined by a type "bandwidth" in the database. Then in SQL, using the *AVG()* command, we compute across these links the average of all the 'bandwidth' events. This now forms a baseline for link bandwidth. SQL can now supply us with all of the dips in the bandwidth over the time period of interest by performing a comparison of bandwidth values against this baseline. Finally, we extract all events within 1 minute of one of these bandwidth dips on both the sending and receiving hosts.

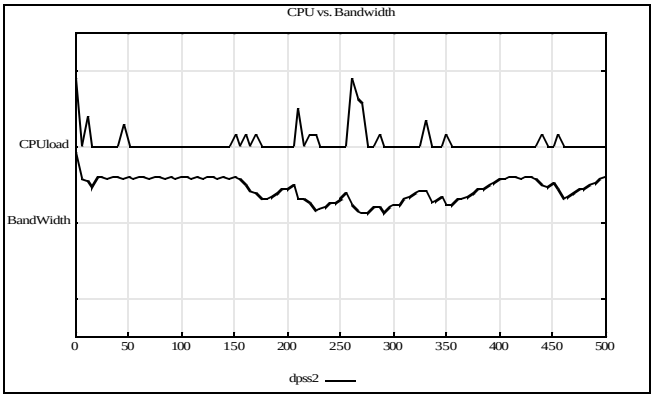

Figure 6: NLV Visualization of Network Loss vs. CPU Load

The results of these queries are graphed with *nlv,* the NetLogger analysis tool, shown in Figure6.This query took only 2.2 seconds to execute on a database with one million events. We have also developed a web browser interface to the archive that facilitates the selection of events, and allows one to send the results to a file or to the *nlv* analysis tool.

The final version of this paper will include more sample queries, results, and performance data on how long the queries take. We are currently building up an archive with more data and more types of monitoring sensors, and should be able to demonstrate much more interesting queries soon.

9.0 Conclusions and Future Work

In this paper we have explained how a relational monitoring event archive is useful for correlating events and tracking down performance issues in distributed systems. A relational database with historical data allows for the establishment of a *baseline* of performance in a distributed environment, and finding events that deviate from this baseline is then a trivial using SQL queries. For the final version of this paper, we will show this monitoring system in a Grid Testbed (such as the DOE Science Grid), and give results for analyzing real Grid applications (such as GridFTP).

10.0 Acknowledgments

We want to thank members of the Global Grid Forum Discovery and Monitoring Event Description Working Group, especially Martin Swany, for there help in defining the data model we are using for monitoring events. This work was supported by the Director, Office of Science. Office of Advanced Scientific Computing Research. Mathematical, Information, and Computational Sciences Division under U.S. Department of Energy Contract No. DE-AC03-76SF00098. This is report no. LBNL-50216.

11.0 References

[1] Allcock B., Bester, J., Bresnahan, J., Chervenak, A., Foster, I., et.al. *Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing*. IEEE Mass Storage Conference, 2001.

[2] Bethel, W., B. Tierney, J. Lee, D. Gunter, S. Lau. *Using High-Speed WANs and Network Data Caches to Enable Remote and Distributed Visualization*. Proceeding of the IEEE Supercomputing 2000 Conference, Nov. 2000.

[3] Cancio, G., S. Fisher, T. Folkes, F. Giacomini, W. Hoschek, D. Kelsey, B. Tierney. The DataGrid Architecture. http://grid-atf.web.cern.ch/grid-atf/doc/architecture-2001-07-02.pdf

[4] Coghlan, B., *A case for Relational GIS/GMA using Relaxed Consistency*, GGF Informational Draft GWD-GP-11-1, http://www.gridforum.org/1_GIS/RDIS.htm

[5] CORBA. Systems Management: Event Management Service. X/Open Document Number: P437, http://www.opengroup.org/onlinepubs/008356299/

[6] Dinda, P. and B. Plale. *A Unified Relational Approach to Grid Information Services*. Grid Forum Informational Draft GWD-GIS-012-1, http://www.gridforum.org/1_GIS/RDIS.htm

[7] European Data Grid Project http://www.eu-datagrid.org/

[8] Fisher, S., *Relational Model for Information and Monitoring*, GGF Informational Draft GWD-GP-7-1, http://www.gridforum.org/1_GIS/RDIS.htm

[9] Fisher, S. Relational Grid Monitoring Architecture Package, http://hepunx.rl.ac.uk/grid/wp3/releases.html

[10] Global Grid Forum (GGF): http://www.globalgridforum.org/

[11] GriPhyN Project: http://www.griphyn.org/

[12] Gunter, D., B. Tierney, K. Jackson, J. Lee, M. Stoufer, *Dynamic Monitoring of High-Performance Distributed Applications*, Proceedings of the 11th IEEE Symposium on High Performance Distributed Computing, July 2002.

[13] Hoschek, W., G. McCance, *Grid Enabled Relational Database Middleware*, Global Grid Forum Informational Draft http://www.gridforum.org/1_GIS/RDIS.htm

[14] Miller, B., Callaghan, M., et al., *The Paradyn parallel performance measurement tools*, IEEE Computer, Vol. 28 (11), Nov. 1995.

[15] Mills, D., *Simple Network Time Protocol (SNTP)*, RFC 1769, University of Delaware, March 1995. http://www.eecis.udel.edu/~ntp/

[16] Particle Physics Data Grid (PPDG): http://www.ppdg.net/

[17] Ribler, R., J. Vetter, H. Simitci, D. Reed. *Autopilot: Adaptive Control of Distributed Applications*. Proceedings of the 7th IEEE Symposium on High-Performance Distributed Computing, Chicago, IL, July 1998.

[18] Smith, W. *A Framework for Control and Observation in Distributed Environments*. NAS Technical Report Number: NAS-01-006, http://www.nas.nasa.gov/~wwsmith/

[19] Snodgrass, R., *A Relational Approach to Monitoring Complex Systems*, ACM Transactions on Computer Systems, Vol. 6, No. 2 (1988), 157-196.

[20] SQL. Database Language SQL. ANSI X3.135-1992

[21] Swany, M. and R. Wolski, *Representing Dynamic Performance Information in Grid Environments with the Network Weather Service*, Proceeding of the 2nd IEEE International Symposium on Cluster Computing and the Grid, Berlin, Germany, May 2002

[22] Thain, D., Jim Basney, Se-Chang Son, Miron Livny. *The Kangaroo Approach to Data Movement on the Grid*. Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing, San Francisco, California, August 2001

[23] Tierney, B., R. Aydt, D. Gunter, W. Smith, V. Taylor, R. Wolski, M. Swany. *A Grid Monitoring Service Architecture*. Global Grid Forum White Paper. http://www-didc.lbl.gov/GridPerf/

[24] Tierney, B., D. Gunter, J. Becla, B. Jacobsen, D. Quarrie. *Using NetLogger for Distributed Systems Performance Analysis of the BaBar Data Analysis System*. Proceedings of Computers in High Energy Physics 2000 (CHEP 2000), Feb. 2000.

[25] Tierney, B., W. Johnston, B. Crowley, G. Hoo, C. Brooks, D. Gunter. *The NetLogger Methodology for High Performance Distributed Systems Performance Analysis*. Proceeding of IEEE High Performance Distributed Computing, July 1998, http://www-didc.lbl.gov/NetLogger/

[26] Wu, X., Taylor, V., et. al., *Design and Development of Prophesy Performance Database for Distributed Scientific Applications*, Proc. the 10th SIAM Conference on Parallel Processing for Scientific Computing, Virginia, March 2001.

[27] Vazhkudai, S., S. Tuecke, I. Foster. *Replica selection in the Globus Data Grid*. International Workshop on Data Models and Databases on Clusters and the Grid (DataGrid 2001), IEEE Computer Society Press, 2001.

[28] Yan, L., Sarukkai, S., and Mehra, P., *Performance measurement, visualization and modeling of parallel and distributed programs using the AIMS toolkit*, Software Practice and Experience, Vol. 25 (4), April 1995.